深度学习与自然语言处理第一次大作业

——中文信息熵的计算

19231178 于天贺

一、实验目标

将给定的金庸的 16 篇小说作为语料库，分别以字、一元词组、二元词组与三元词组进行中文信息熵的计算。

二、实验原理

2.1. 信息熵

信息熵是信息论中一个非常重要的概念，它可以用来衡量随机变量的不确定度或信息量。熵的概念最初是由克劳德香农在 1948 年提出的，它被广泛应用于通信、密码学、数据压缩、信号处理等领域。

在信息论中，熵的定义是一个离散随机变量的平均信息量。它可以被看作是描述随机事件的不确定性的量度。

对于一个离散型随机变量 X，其信息熵的定义为

$$H(X)=-\sum_i^n P(x_i)\log P(x_i)$$

2.2. 统计语言模型

假定 S 表示某个有意义的句子，由一连串特定顺序排列的字或者词ω1 , ω2 , ω3 , . . . ωn 组成这里 n 是组成句子的字或词的数量。现在我们想知道 s 在文本中出现的可能性，即：

$$P(s) = P(\omega1 , \omega2 , \omega3 , . . . \omega n)$$

利用条件概率公式可以得到：

$$P(\omega1 , \omega2 , \omega3 , . . . \omega n) = P(\omega1)P(\omega2 |\omega1). . . P(\omega n |\omega1 , \omega2 , . . . \omega n-1)$$

下面以词为例（字与其原理相同），进行理论分析。其中 P(ω1)表示第一个词ω1 出现的概率；P(ω2|ω1)是在已知第一个词的前提下，第二个词出现的概率，以此类推 P(ωn |ω1 , ω2 , . . . ωn-1)是在已知第 1,2,3...n-1 个词的前提下，第 n 个词出现的概率。当计算 P(ω1)时，仅存在一个参数；计算 P(ω1 |ω2)，存在两个参数，以此类推，在句子又多又长的情况下，难易计算。

所以马尔可夫提出一种假设：假设 N 元模型的每个词出现的概率只与前面 N-1 个词相关，当 N=2 时，就是二元模型，N=3 就是三元模型。N 元模型，即当前这个词ωi 依赖于前面 N-1 个词，上述 N=1 为与前面单词都没有关系，N=2 表示与前面一个单词有关；N=3 表示与前面两个词有关。选取不同模型最终结果也会不同。

当 N=1 时，一元模型的数学表达如下所示：

$$P(s)=P(w1)P(w2)P(w3)...P(wi)...P(wn)$$

而当 N=2 时，二元模型的数学表达如下所示：

$$P(s)=P(\omega1)P(\omega2|\omega1)P(\omega3|\omega2)...P(\omega i|\omega i-1)...P(\omega n|\omega n-1)$$

而当 N=3 时，三元模型的数学表达如下所示：

$$P(s)=P(w1)P(w2|w1)P(w3|w1,w2)...P(wi|wi-2,wi-1)...P(wn|wn-2,wn-1)$$

2.3. 计算语言模型的信息熵

如果统计量足够，字、词、二元词组或三元词组出现的概率大致等于其在语料库中出现的频率。

下面以词为例（字与其原理相同），进行理论分析。

一元模型的信息熵计算公式如下

$$H(X) = -\sum P(x) log P(x)$$

其中 P(x)可近似等于每个词组在语料库中出现的频率。

二元模型的信息熵计算公式如下

$$H(X|Y) = -\sum P(x,y) log P(x|y)$$

其中联合概率 P(x,y)可近似等于每个二元词组在语料库中出现的频率，条件概率 P(x|y)可近似等于每个二元词组在语料库中出现的频数与以该二元词组的第一个词为词首的二元词组的频数的比值。

三元模型的信息熵计算公式如下

$$H(X|Y,Z) = -\sum P(x,y,z) log P(x|y,z)$$

其中联合概率 P(x,y,z)可近似等于每个三元词组在语料库中出现的频率，条件概率 P(x|y,z)可近似等于每个三元词组在语料库中出现的频数与以该三元词组的前两个词为词首的三元词组的频数的比值。

三、实验过程

3.1 数据的装载与预处理

通过面向对象编程的方法，自定义一个读取数据与数据预处理的类

```
class ReadFile:
    def __init__(self, root_dir):
        self.root_dir = root_dir

    def get_corpus(self):
        text_list = []
        r1 = '[a-zA-Z0-9’!"#$%&\'()*+,-./:: ;<=>?@，。?★、…【】《》？“”‘’！[\\]^_`{|}~]+'
        listdir = os.listdir(self.root_dir)
        characters_count = 0
        for file_name in listdir:
            path = os.path.join(self.root_dir, file_name)
            if os.path.isfile(path) and file_name.split('.')[-1] == 'txt':
                with open(os.path.abspath(path), "r", encoding='ansi') as file:
                    print('file:', file)
                    file_content = file.read()
                    file_content = re.sub(r1, '', file_content)
                    file_content = file_content.replace("\n", '')
                    file_content = file_content.replace(" ", '')
                    file_content = file_content.replace('\u3000', '')
                    file_content = file_content.\
                        replace("本书来自www.cr173.com免费txt小说下载站\n更多更新免费电子书请关注www.cr173.com", '')
                    characters_count += len(file_content)
                    text_list.append(file_content)
            elif os.path.isdir(path):
                print('文件路径不存在!!!!!!')
        return text_list, characters_count
```

所有语料库资料都在 txt 文件中，所以首先排除非 txt 文件。

之后，由于 txt 文件中都是中文字符，编码格式 encoding 选择了很多，例如 gbk 等等，都报错，只有 ansi 不报错，可以用来读取。

之后利用正则表达式，将 txt 文件中的小说里面所有无关的字符，例如标点符号，数字，英文字母等等全部去掉。

之后通过 string.replace()方法，去掉所有换行符与空格，但在后面代码调试

时发现，没有把中文全角空格字符删除，即\u3000，于是继续删除\u3000。

之后去除广告。

最后把处理完的每一篇文本（字符串类型数据），保存到列表中，并返回，同时返回所有 16 篇小说的总字数。

3.2. 计算基于字分割的一元模型的信息熵

代码实现如下

```
def calculate_single_character_entropy(corpus):
    before = time.time()
    split_characters = []
    for text in corpus:
        split_characters += [char for char in text]
    characters_count = len(split_characters)
    single_character_dict = dict(Counter(split_characters))
    print("语料库字数:", characters_count)

    entropy = []
    for _, value in single_character_dict.items():
        entropy.append(-(value / characters_count) * math.log(value / characters_count, 2))
    print("基于字的中文信息熵为:", round(sum(entropy), 5))
    after = time.time()
    print("运行时间:", round(after - before, 5), "s")
```

基本思路是，先对存储 16 篇小说的列表进行处理，将 16 篇小说组成的语料库中的所有字，存储在一个列表 split_characters 中，这个列表中的元素可能有重复（相同字），此时需要统计每个字在列表中出现的频数，并保存为字典，这时就利用 python 自带的 Counter 方法，得到了键为每个字，值为该字的频数的字典。之后再通过公式计算总的信息熵即可。

3.3. 计算基于词组分割的一元模型的信息熵

代码实现如下

```
def calculate_unary_entropy(corpus, characters_count):
    before = time.time()
    split_words = []
    for text in corpus:
        split_words += list(jieba.cut(text))
    words_count = len(split_words)
    unary_words_dict = dict(Counter(split_words))

    print("语料库字数:", characters_count)
    print("一元词组数量:", words_count)
    print("平均词长:", round(characters_count / words_count, 5))

    entropy = []
    for _, value in unary_words_dict.items():
        entropy.append(-(value / words_count) * math.log(value / words_count, 2))
    print("基于词的一元模型的中文信息熵为:", round(sum(entropy), 5))
    after = time.time()
    print("运行时间:", round(after - before, 5), "s")
```

基本思路与 3.2 基本相同，先对存储 16 篇小说的列表进行处理，不过需要使用 jieba 库来对文本进行分词，将 16 篇小说组成的语料库中的所有词，存储在一个列表 split_words 中，这个列表中的元素可能有重复（相同词），此时需要统计每个词在列表中出现的频数，并保存为字典，这时就利用 python 自带的 Counter 方法，得到了键为每个词，值为该词的频数的字典。之后再通过公式计算总的信息熵即可。

3.4. 计算基于词组分割的二元模型的信息熵

代码实现如下

```
def calculate_binary_entropy(corpus, characters_count):
    before = time.time()
    all_words = []
    binary_words_dict = {}
    for text in corpus:
        split_words = list(jieba.cut(text))
        for i in range(len(split_words) - 1):
            binary_words_dict[(split_words[i], split_words[i + 1])] = binary_words_dict.get(
                (split_words[i], split_words[i + 1]), 0) + 1
        all_words += split_words

    words_count = len(all_words)
    unary_words_dict = dict(Counter(all_words))

    print("语料库字数:", characters_count)
    print("一元词组数量:", words_count)
    print("平均词长:", round(count / words_count, 5))

    binary_words_count = sum([value for _, value in binary_words_dict.items()])
    print("二元词组数量:", binary_words_count)

    entropy = []
    for key, value in binary_words_dict.items():
        joint_probability_xy = value / binary_words_count  # 计算联合概率p(x,y)
        conditional_probability_x_y = joint_probability_xy / (unary_words_dict[key[0]]/words_count)  # 计算条件概率p(x|y)
        entropy.append(-joint_probability_xy * math.log(conditional_probability_x_y, 2))  # 计算二元模型的信息熵
    print("基于词的二元模型的中文信息熵为:", round(sum(entropy), 5))

    after = time.time()
    print("运行时间:", round(after - before, 5), "s")
```

计算二元模型的信息熵，需要既知道每个二元词组的联合概率 p(x,y)，又需要知道每个二元词组中以前一个词为条件，后一个词出现的条件概率 p(x|y)。所以，需要一个二元词组字典 binary_words_dict 存储每个二元词组在语料库中出现的次数。同时计算条件概率时，仍然需要一元词字典 unary_words_dict，计算每个二元词组中的前一个词的出现的概率 p(y)，然后通过 p(x|y)=p(x,y)/p(y)计算条件概率。

3.5. 计算基于词组分割的三元模型的信息熵

代码实现如下

```
def calculate_ternary_entropy(corpus, characters_count):
    before = time.time()
    all_words = []
    binary_words_dict = {}
    ternary_words_dict = {}
    for text in corpus:
        split_words = list(jieba.cut(text))
        for i in range(len(split_words) - 1):
            binary_words_dict[(split_words[i], split_words[i + 1])] = binary_words_dict.get(
                (split_words[i], split_words[i + 1]), 0) + 1
        for i in range(len(split_words) - 2):
            ternary_words_dict[((split_words[i], split_words[i + 1]), split_words[i + 2])] = ternary_words_dict.get(
                ((split_words[i], split_words[i + 1]), split_words[i + 2]), 0) + 1
        all_words += split_words
    words_count = len(all_words)
    unary_words_dict = dict(Counter(all_words))

    print("语料库字数:", characters_count)
    print("一元词组数量:", words_count)
    print("平均词长:", round(characters_count / words_count, 5))

    binary_words_count = sum([value for _, value in binary_words_dict.items()])
    print("二元词组数量:", binary_words_count)
    ternary_words_count = sum([value for _, value in ternary_words_dict.items()])
    print("三元词组数量:", ternary_words_count)

    entropy = []
    for key, value in ternary_words_dict.items():
        joint_probability_xyz = value / ternary_words_count  # 计算联合概率p(x,y,z)
        conditional_probability_x_yz = joint_probability_xyz / (binary_words_dict[key[0]] / binary_words_count)  # 计
        entropy.append(-joint_probability_xyz * math.log(conditional_probability_x_yz, 2))  # 计算三元模型的信息熵
    print("基于词的三元模型的中文信息熵为:", round(sum(entropy), 5))

    after = time.time()
    print("运行时间:", round(after - before, 5), "s")
```

思路与计算二元模型的信息熵类似，计算三元模型信息熵，需要知道每个三元词组在语料库出现的频数，来计算联合概率 p(x,y,z)，所以需要 ternary_words_dict 存储每个三元词组在语料库中出现的次数。同时又需要知道每个三元词组中以前两个词构成的二元词组出现的频数来计算得到以前两个词为条件，后一个词出现的条件概率 p(x|y,z)，所以仍然需要二元词字典 binary_words_dict，计算每个三元词组中的前两个词的出现的概率 p(y,z)，最后通过 p(x|y,z)=p(x,y,z)/p(y,z)得到条件概率。

3.6. 执行主函数得到结果

```
if __name__ == '__main__':
    read_file = ReadFile("./jyxstxtqj_downcc.com")
    text_list, count = read_file.get_corpus()

    calculate_single_character_entropy(corpus=text_list)
    calculate_unary_entropy(corpus=text_list, characters_count=count)
    calculate_binary_entropy(corpus=text_list, characters_count=count)
    calculate_ternary_entropy(corpus=text_list, characters_count=count)
```

四、实验结果

基于单个字模型的语料库信息熵为 9.53897，代码运行结果如下

```
语料库字数: 7296203
基于字的中文信息熵为: 9.53897
运行时间: 2.34891 s
```

基于一元词组模型的语料库信息熵为 12.16803，代码运行结果如下

```
语料库字数: 7296203
一元词组数量: 4302558
平均词长: 1.69578
基于词的一元模型的中文信息熵为: 12.16803
运行时间: 73.84046 s
```

基于二元词组模型的语料库信息熵为 6.94001，代码运行结果如下

```
语料库字数: 7296203
一元词组数量: 4302558
平均词长: 1.69578
二元词组数量: 4302541
基于词的二元模型的中文信息熵为: 6.94001
运行时间: 79.46062 s
```

基于三元词组模型的语料库信息熵为 2.31152，代码运行结果如下

```
语料库字数: 7296203
一元词组数量: 4302558
平均词长: 1.69578
二元词组数量: 4302541
三元词组数量: 4302524
基于词的三元模型的中文信息熵为: 2.31152
运行时间: 102.20644 s
```

五、额外实验

停词对文章的信息熵有一定的影响。停词是指在文本中出现频率非常高的一些词，这些词往往不具有明确的语义，例如“的”、“了”、“是”等。这些词如果不进行处理，在计算信息熵时会对整个文本的信息熵产生影响，因为它们的出现概率很高，而它们并不携带过多的信息。

因此，在计算信息熵时，通常需要将停词去除，以减少其对结果的影响。这样可以更好地反映文本中实际含有的信息量，使得信息熵更加准确。

删除停词函数如下

```python
def delete_stop_words(stop_words_path, corpus):
    before = time.time()
    with open(stop_words_path, 'r', encoding='utf-8') as stop_words_file:
        stop_words = [line.strip() for line in stop_words_file.readlines()]

    new_corpus = []
    character_count = 0
    for text in corpus:
        new_words = []
        split_words = list(jieba.cut(text))
        for word in split_words:
            if word not in stop_words:
                new_words.append(word)
        character_count += len(''.join(map(str, new_words)))
        new_corpus.append(''.join(map(str, new_words)))

    after = time.time()
    print("删除停词运行时间:", round(after - before, 5), "s")
    return new_corpus, character_count
```

删除停词后，基于单个字模型的语料库信息熵为 9.85，代码运行结果如下

```
语料库字数: 5650166
基于字的中文信息熵为: 9.85
运行时间: 1.81058 s
```

删除停词后，基于一元词组模型的语料库信息熵为 13.77197，代码运行结果如下

```
语料库字数: 5650166
一元词组数量: 2819193
平均词长: 2.00418
基于词的一元模型的中文信息熵为: 13.77197
运行时间: 49.92975 s
```

删除停词后，基于二元词组模型的语料库信息熵为 6.41183，代码运行结果如下

```
语料库字数: 5650166
一元词组数量: 2819193
平均词长: 2.58805
二元词组数量: 2819176
基于词的二元模型的中文信息熵为: 6.41183
运行时间: 57.6817 s
```

删除停词后，基于三元词组模型的语料库信息熵为 1.10362，代码运行结果如下

六、实验结果分析

通过对比字和一元词的信息熵可以发现，一元词的大于字的，基于字模型的信息熵要低于基于一元词模型的信息熵，主要有以下几个原因：
字的种类比词的种类要少。汉字的种类有几千种，而常用的字不到一千个。相比之下，汉语的词汇量非常庞大，达到了数十万或者更多，所以基于一元词模型的信息熵会更大。

字在不同的上下文中具有更加确定的含义。相对于词而言，字可以更加准确地表示某个概念。因为在不同的上下文中，同一个词可能具有不同的含义，而同一个字在不同的上下文中一般来说只有一个含义。

基于字模型更容易识别新词。如果在使用基于词模型的语言模型时遇到了新词，模型就会无法处理。但是基于字模型的语言模型在遇到新词时可以通过组合已有的字来表示新词，因此更加灵活和准确。

而二元模型的信息熵要低于一元模型，是因为在二元模型中，每个字符的出现概率与其前一个字符有关。因此，二元模型在语言建模中能够更好地捕捉到字符之间的关系，使得相邻字符之间的信息冗余量减少，从而降低了信息熵。而在一元模型中，每个字符的出现概率只与该字符本身有关，不能捕捉到字符之间的关系，因此相邻字符之间的信息冗余量较大，导致信息熵较高。三元模型信息熵要低于二元模型的原因与之同理。

通过额外实验发现，将停词删除后，所有一元模型信息熵均有所提升，而二元模型与三元模型的信息熵降低。因为删除停词后，文本中的冗余信息被剔除，而剩余的有效信息更加集中。在二元模型和三元模型中，每个元素都是以连续两个或三个词为单位进行统计的，因此相比于一元模型，它们更能反映词汇之间的语义关系和上下文信息。因此，当停词被移除后，二元模型和三元模型能够更准确地反映文本中词汇的相关性和上下文语境，导致它们的信息熵更低。